A linear-time algorithm for Euclidean feature transform sets

Hesselink, Wim H.

Published in:
Information Processing Letters

DOI:
10.1016/j.ipl.2006.12.005

Publication date:
2007

Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 102 (2007) 181–186
Information Processing Letters
www.elsevier.com/locate/ipl

A linear-time algorithm for Euclidean feature transform sets

Wim H. Hesselink

Department of Mathematics and Computing Science, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands

Received 27 May 2005

Available online 19 December 2006

Communicated by F. Dehne

Abstract

The Euclidean distance transform of a binary image is the function that assigns to every pixel the Euclidean distance to the background. The Euclidean feature transform is the function that assigns to every pixel the set of background pixels with this distance. We present an algorithm to compute the exact Euclidean feature transform sets in linear time. The algorithm is applicable in arbitrary dimensions.

Keywords: Algorithms; Distance transform; Feature transform; Image processing; Computational geometry

1. Introduction

The concept of distance transformation was introduced in [11]. Given a binary image, the distance transform $dt(x)$ of a grid point x is the smallest distance of x to any point of the background. For many years, people have been satisfied with so-called chamfer distances, see [1]. In 1980, Danielson [5] presented two efficient algorithms that give good approximations of the Euclidean distance. The first exact Euclidean distance transform algorithms in linear time were given in [3,2,7]. Subsequently, other versions were invented by ourselves [9] and Maurer et al. [8].

Feature transform is an abbreviation of nearest feature transform [10]. There are two versions. The *Euclidean feature transform set* $FT(x)$ of a grid point x consists of the set of background pixels with minimal Euclidean distance to x. The *simple feature transform* $ft(x)$ just yields one element of $FT(x)$, say the first element in some lexical ordering of the pixels.

In [6], we presented an algorithm to compute a simple feature transform in linear time. This was a rather straightforward extension of the algorithm of [9] for the Euclidean distance transform. We used the simple feature transform to define the so-called integer medial axis (*IMA*), which is a kind of skeleton that can be computed in linear time. Indeed, for many purposes, the simple feature transform seems to be good enough. Yet, its use introduces a disturbing kind of nondeterminism.

The paper [4] uses feature transform sets to determine Euclidean skeletons of images. It calls the feature transform set the *downstream*. The algorithm of [4] to compute the (extended) downstream uses the Euclidean distance transform and a lookup table for the integral vectors of given lengths. For 2D, the number of integral vectors with a given square length n is approximately constant (the average is π). In dimension $d > 2$, however, the number grows as $n^{(d-2)/2}$. It follows that the algorithm of [4] is not linear-time in higher dimensions.

E-mail address: w.h.hesselink@rug.nl (W.H. Hesselink).
URL: http://www.cs.rug.nl/~wim.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2006.12.005

In 2D, we also prefer to avoid construction and inspection of lookup tables whenever possible.

In the present paper, we develop an algorithm for the Euclidean feature transform sets as an extension of the derivation of the algorithm for the Euclidean distance transform. It is based on induction on the dimension, while the induction step requires us to develop the so-called concave Minarg algorithm.

Given a function f of two integer arguments, the Minarg problem asks to compute, as a function of y, the sets of arguments z for which $f(y,z)$ is minimal. The concave Minarg algorithm solves this problem efficiently for the case that function f is what we call *concave*. Although developed to compute feature transform sets, this algorithm may well be applicable to other problem areas.

Overview. Section 2 contains the main definitions, the one-dimensional case, the reduction to the Minarg algorithm, and the analysis of concavity. Section 3 contains the derivation of the concave Minarg algorithm. Section 4 contains concluding remarks.

2. The general algorithm

We first set the stage for binary images in arbitrary dimension d (though usually $d = 2$ or 3). The grid $\mathbb{Z}^d$ is regarded as a subset of the Euclidean vector space $\mathbb{R}^d$ with its standard length function. For grid points x and y, the squared distance is $\|x-y\|^2 = \sum_i (x_i - y_i)^2$. This is always a natural number. Computation of the square root is usually superfluous.

We define an *image* to be a pair (A, X) where A is a rectangular box in $\mathbb{Z}^d$ and X is a subset of A which is called the *foreground*. The complement B of X within A is called the *background*. So we have $B = A \setminus X$.

For every $x \in A$, we define the *distance transform* $dt(x, B)$ as the squared minimum distance of x to any point of the background B. The feature transform set $FT(x, B)$ of x is defined to consist of the background points y at a squared distance equal to $dt(x, B)$. To avoid empty feature transforms and infinite distances, we add some background points to B that are far away from the box A (thus, for convenience, we allow $B \not\subseteq A$). When B is fixed, we usually suppress the argument B and simply write $dt(x)$ and $FT(x)$. So we have $FT(x) = \{y \in B \mid dt(x) = \|x-y\|^2\}$.

2.1. The one-dimensional case

The aim is to compute $FT(x)$ for all $x \in A$. The basic idea is to use induction in the dimension.

The base case has dimension $d = 1$. In this case, A is a grid line and we may assume that it consists of the integers x with $0 \leqslant x < m$ for fixed m. So $A = [0 \ldots m)$. The algorithm consists of two scans. In *FirstScan*, it initializes an array `ri` which gets the distance to the closest right-hand background point. Since initially no background points have been observed, we use some number $large > m$ for initialization. This effectively adds some far away background points.

$w := large$;
for $x := m-1$ **downto** 0 **do**
  $w := (x \in B \,?\, 0 : w+1)$;
  $\mathtt{ri}[x] := w$;
end $\{w = \mathtt{ri}[0]\}$

Fragment *SecondScan* reuses w as a memory of the previous position.

for $x := 0$ **to** $m-1$ **do**
  if $x - w = \mathtt{ri}[x] \neq 0$ **then**
    $FT(x) := \{w, x + \mathtt{ri}[x]\}$;
    $w := x + \mathtt{ri}[x]$;
  elsif $x - w < \mathtt{ri}[x]$ **then** $FT(x) := \{w\}$
  else
    $w := x + \mathtt{ri}[x]$;
    $FT(x) := \{w\}$;
  end
end.

The only subtle point is in the first alternative where $FT(x)$ becomes a set with two elements since the closest background points to the left and to the right are equally close. The condition $\mathtt{ri}[x] \neq 0$ guarantees that these points differ. We include this test to implement the sets as lists without multiple occurrences.

2.2. The induction step

In the induction step, we assume that $d > 1$ and that algorithms to compute dt and FT in dimension $d-1$ are available. We then have to compute dt and FT in dimension d. The rectangular box A is now a Cartesian product of the form $A = A' \times [0 \ldots n)$ where A' is a rectangular box in $\mathbb{Z}^{d-1}$ and $n \in \mathbb{N}$. The background B is a union of n slices $B_z \times \{z\}$ with $0 \leqslant z < n$, where $B_z = \{q \in \mathbb{Z}^{d-1} \mid (q, z) \in B\}$.

For every $p \in A'$, we shall compute $FT(x)$ for all grid points x on the line of the grid points (p, y) with

$0 \leqslant y < n$. So, we let $p \in A'$ be fixed. By the theorem of Pythagoras, the distance transform $dt((p, y), B)$ of a grid point (p, y) is the minimum value of $f(y, z) = (y-z)^2 + h(z)$, where z ranges over $[0 \ldots n)$ and $h(z) = dt(p, B_z)$.

Now, $h(z)$ can be computed since it is a lower-dimensional distance transform. Let $\mathrm{Minarg}_y(f)$ be the set of arguments z where $f(y, z)$ is minimal. Then the feature transform $FT((p, y), B)$ of (p, y) is the set of grid points (q, z) with $z \in \mathrm{Minarg}_y(f)$ and $q \in FT(p, B_z)\}$.

Since the lower-dimensional feature transform $FT(p, B_z)$ can be computed by assumption, the problem is to compute the sets $\mathrm{Minarg}_y(f)$.

2.3. Concavity

In the next section we derive an efficient algorithm for $\mathrm{Minarg}_y(f)$. We do this by generalizing the problem. Function f is a numerical function of two integer arguments with one special property. It satisfies $f(p, y) + f(q, z) < f(p, z) + f(q, y)$ whenever $p < q$ and $y < z$. We call this property *concavity*, although we would prefer a more appropriate name. The property is verified in

$$\begin{aligned}
& f(p, y) + f(q, z) < f(p, z) + f(q, y) \\
\equiv\ & (p-y)^2 + h(y) + (q-z)^2 + h(z) \\
& \quad < (p-z)^2 + h(z) + (q-y)^2 + h(y) \\
\equiv\ & -2(p-q)(y-z) < 0 \\
\Leftarrow\ & p < q \wedge y < z.
\end{aligned}$$

Strictly speaking, we need even less, namely the following immediate consequence:

$$p < q \wedge y < z \wedge f(q, y) \leqslant f(q, z) \quad \Rightarrow \quad f(p, y) < f(p, z). \tag{0}$$

This formula implies that, for given y and z with $y < z$, the set of arguments p with $f(p, y) \leqslant f(p, z)$ is an initial segment. In general, the end point of this segment can be computed by binary search. In the specific case of our function f, however, the end point can be computed in constant time because of

$$\begin{aligned}
& f(p, y) \leqslant f(p, z) \\
\equiv\ & (p-y)^2 + h(y) \leqslant (p-z)^2 + h(z) \\
\equiv\ & 2p(z-y) \leqslant z^2 - y^2 + h(z) - h(y) \\
\equiv\ & \{\ \text{assume } y < z\ \} \\
& p \leqslant \big(z^2 - y^2 + h(z) - h(y)\big)/\big(2(z-y)\big).
\end{aligned}$$

We therefore define the *separator function* g by

$$g(y, z) = \Big\lfloor \big(z^2 - y^2 + h(z) - h(y)\big)/\big(2(z-y)\big) \Big\rfloor.$$

If $y < z$, this function g satisfies for all $p \in [0 \ldots n)$:

$$f(p, y) \leqslant f(p, z) \quad \equiv \quad p \leqslant g(y, z). \tag{1}$$

Notice, however, that $g(y, z)$ is not bounded to $[0 \ldots n)$.

3. The concave Minarg algorithm

In the remainder of the paper, we concentrate on the computation of $\mathrm{Minarg}_y(f)$ for a numerical function f of two natural arguments that satisfies the formulas (0) and (1). Formula (1) is not used until Section 3.2.

3.1. Problem analysis

For the development of the algorithm, we keep the range $[0 \ldots n)$ of y constant but replace the range of z by $[0 \ldots k)$ where k is a variable. Now $\mathrm{Minarg}_y(f) = M(y, n)$ where the function M of two integer variables is given by

$$M(y, k) = \big\{ z \in [0 \ldots k) \mid \forall\, x \in [0 \ldots k): f(y, z) \leqslant f(y, x) \big\}. \tag{2}$$

We obviously have $\emptyset \neq M(y, k) \subseteq [0 \ldots k)$ for all $k \geqslant 1$ and all y. In particular, $M(y, 1) = \{0\}$. With respect to incrementation of k, the definition of function M implies that $M(y, k+1) = M(y, k)$ if $f(y, k)$ is bigger than the minimal value of $f(y, \_)$ on $[0 \ldots k)$, that $M(y, k+1) = \{k\}$ if $f(y, k)$ is less than this minimal value, and that $M(y, k+1) = M(y, k) \cup \{k\}$ in the case of equality. This shows that, for every $z \in M(y, k)$, we have

$$\begin{aligned}
& M(y, k+1) = \\
& \quad \textbf{if}\ f(y, z) < f(y, k)\ \textbf{then}\ M(y, k) \\
& \quad \textbf{elsif}\ f(y, z) = f(y, k)\ \textbf{then}\ M(y, k) \cup \{k\} \\
& \quad \textbf{else}\ \{k\}\ \textbf{end}.
\end{aligned} \tag{3}$$

This recurrence relation leads to a straightforward algorithm for the computation of the sets $M(\_, k)$, which is of order $\mathcal{O}(n \cdot k)$.

To get a more efficient algorithm, we use that f satisfies formula (0). Together with definition (2), this implies that function M is in some sense monotonic in its first argument:

$$p < q \wedge y \in M(p, k) \wedge z \in M(q, k) \quad \Rightarrow \quad y \leqslant z. \tag{4}$$

This is proved in

$$
\begin{array}{ll}
& p < q \wedge z < y \wedge y \in M(p,k) \wedge z \in M(q,k) \\
\Rightarrow & \{\ (2) \text{ with } y := q \text{ and } x := y\ \} \\
& p < q \wedge z < y \wedge y \in M(p,k) \wedge f(q,z) \leqslant f(q,y) \\
\Rightarrow & \{\ (0)\ \} \\
& z < y \wedge y \in M(p,k) \wedge f(p,z) < f(p,y) \\
\Rightarrow & \{\ (2) \text{ with } y := p, z := y, x := z \text{ gives} \\
& \quad f(p,y) \leqslant f(p,z)\ \} \\
& \mathit{false}.
\end{array}
$$

For $k > 0$, we define $a(p,k)$ and $b(p,k)$ to be the minimum and maximum of $M(p,k)$, respectively. Formula (4) now implies

$$p < q \quad \Rightarrow \quad b(p,k) \leqslant a(q,k). \tag{5}$$

Formula (4) also implies that both functions $a(p,k)$ and $b(p,k)$ are ascending in p. It follows that the set of arguments p with $a(p,k+1) = k$ is a final (possibly empty) segment of the range of p, and similarly for $b(p,k+1) = k$. We therefore define the functions u and v as the starting points of these final segments:

$$
\begin{aligned}
u(k) &= \min\bigl\{p \mid p = n \vee a(p,k+1) = k\bigr\}, \\
v(k) &= \min\bigl\{p \mid p = n \vee b(p,k+1) = k\bigr\}.
\end{aligned}
$$

We now have

$$
\begin{array}{ll}
& p < u(k) \\
\equiv & a(p,k+1) < k \\
\equiv & \forall\, z \in [0 \ldots k+1)\colon\ f\bigl(p, a(p,k)\bigr) \leqslant f(p,z) \\
\equiv & f\bigl(p, a(p,k)\bigr) \leqslant f(p,k).
\end{array} \tag{6}
$$

For function v, a different calculation yields a similar result:

$$
\begin{array}{ll}
& p < v(k) \\
\equiv & b(p,k+1) < k \\
\equiv & f\bigl(p, b(p,k)\bigr) < f(p,k) \\
\equiv & f\bigl(p, a(p,k)\bigr) < f(p,k).
\end{array} \tag{7}
$$

It follows that $v(k) \leqslant u(k)$. We claim that $v(k)$ and $u(k)$ cannot differ more than 1. Indeed, if $p = v(k)$ then $b(p,k+1) = k$, so that formula (5) implies $a(p+1, k+1) = k$ and hence $u(k) \leqslant p+1$. This proves

$$v(k) \leqslant u(k) \leqslant v(k) + 1.$$

Since $a(y,k) \in M(y,k)$, it follows from (6) and (7), that formula (3) reduces to

$$
\begin{array}{l}
M(y,k+1) = \\
\quad \textbf{if}\ \ y < v(k)\ \ \textbf{then}\ \ M(y,k) \\
\quad \textbf{elsif}\ \ y < u(k)\ \ \textbf{then}\ \ M(y,k) \cup \{k\} \\
\quad \textbf{else}\ \ \{k\}\ \ \textbf{end}.
\end{array} \tag{8}
$$

By induction in k, this implies that

$$
\begin{aligned}
&M(y,k) = \{i\} \cup \bigl\{j \mid i < j < k \wedge v(j) = y\bigr\}, \\
&\quad \text{where } i = \max\bigl\{j \mid u(j) \leqslant y\bigr\}.
\end{aligned} \tag{9}
$$

The functions u and v thus hold the key to the computation of function M.

3.2. Algorithm design

Since (6) and (7) determine $u(k)$ and $v(k)$ in terms of $a(\_,k)$, we turn to the inductive computation of $a(\_,k)$. Since the sequence $a(\_,k)$ is ascending, we are especially interested in the indices where it increases. Let $q(k)$ be the number of indices where $a(\_,k)$ increases and let $t(1,k)$ up to $t(q(k),k)$ be these indices, in increasing order. We also define $t(0,k) = 0$ and $t(q(k)+1,k) = n$. Then we have

$$
\begin{aligned}
&0 \leqslant i \leqslant q(k) \quad \Rightarrow \quad t(i,k) < t(i+1,k), \\
&0 \leqslant i < q(k) \quad \Rightarrow \quad a\bigl(t(i),k\bigr) < a\bigl(t(i+1),k\bigr), \\
&0 \leqslant i \leqslant q(k) \wedge t(i,k) \leqslant y < t(i+1,k) \\
&\quad \Rightarrow \quad a(y,k) = a\bigl(t(i,k),k\bigr).
\end{aligned} \tag{10}
$$

In the base case, we have $q(1) = 0$, and $a(y,1) = 0$ for all y.

Assume that, for given $k \geqslant 1$, we know $q(k)$ and the numbers $t(i,k)$, and $a(t(i,k),k)$ for $i \leqslant q(k)$. Let us write $s(i,k) = f(t(i,k), a(t(i,k),k))$. Then (6) implies

$$t(i,k) < u(k) \quad \equiv \quad s(i,k) \leqslant f\bigl(t(i,k),k\bigr). \tag{11}$$

One can use (11) to determine whether $u(k) = 0$ or to determine the greatest index j with $t(j,k) < u(k)$, say by linear search.

If $u(k) \neq 0$, we have $t(j,k) < u(k) \leqslant t(j+1,k)$. Using the formulas (6), (10), and (1), we get, for all p with $t(j,k) \leqslant p < t(j+1,k)$, that we have $p < u(k)$ if and only if $p \leqslant g(a(t(j,k),k),k)$. This implies

$$u(k) \leqslant 1 + g\bigl(a\bigl(t(j,k),k\bigr),k\bigr).$$

If $u(k) < n$, formula (6) with $p := u(k)$ implies that $f(u(k),k) < f(u(k), a(u(k),k))$. By the definition of a and M, this implies that $f(u(k),k) < f(u(k),y)$ for every $y < k$. Therefore formula (1) with $p := u(k)$ and $z := k$ implies that $g(y,k) < u(k)$ for all $y < k$. Since $a(t(j,k),k) < k$, it follows that $u(k) = 1 + g(a(t(j,k),k),k)$.

In any case, we have $u(k) \leqslant n$ by definition. We thus get

$$u(k) = \min\big(n, 1 + g\big(a\big(t(j,k),k\big),k\big)\big).$$

Putting $p = u(k) - 1$, we can decide whether $v(k) = u(k)$ or $v(k) = p$ by means of (6) and (7), which imply the equivalence

$$v(k) = p \quad \equiv \quad f\big(p, a\big(t(j,k),k\big)\big) = f(p,k).$$

Now that we have determined $u(k)$ and $v(k)$, we can proceed to determine $q(k')$ and the numbers $t(i,k')$ and $a(t(i,k'),k')$ for $i \leqslant q(k')$ where $k' = k+1$.

For $y < u(k)$, it follows from (8) that $a(y,k') = a(y,k)$, so that the data remain unchanged. On the other hand, for $u(k) \leqslant y < n$, we have $a(y,k') = k$. Therefore $u(k)$ is the last index in the sequence $t(\_,k')$ before n and we have $a(u(k),k') = k$. In principle, this concludes the algorithmic analysis.

3.3. Data representation

It remains to encode the algorithm presented above in such a way that we can efficiently gather the sets $M(y,n)$ for all y. We first declare program variables to compute the data in (10).

$\mathtt{t}$: **array** $[n+1]$ **of** *Integer*;

$\mathtt{k}, \mathtt{q}$: *Integer*;

$\mathtt{at}$: **array** $[n]$ **of** *integer*.

We let $\mathtt{q}$ stand for $q(\mathtt{k})$ and, similarly, preserve the invariants $\mathtt{t}[i] = t(i,\mathtt{k})$ for all $0 \leqslant i \leqslant \mathtt{q}+1$ and $\mathtt{at}[i] = a(\mathtt{t}[i],\mathtt{k})$.

For the representation of M, we observe that (5) and (10) imply that $M(y,k) = \{a(t(i,k),k)\}$ for all y with $t(i,k) \leqslant y < t(i+1,k) - 1$. Therefore, if $M(y,k)$ is not a singleton, we have $y = t(i+1,k) - 1$ for some $i \leqslant q(k)$. Let us represent these sets by means of an array $\mathtt{mt}$ of sets with, for all $i \leqslant \mathtt{q}$, the invariant

$$M\big(\mathtt{t}[i+1]-1, \mathtt{k}\big) = \mathtt{mt}[i].$$

The above analysis leads to the following algorithm that preserves these invariants:

$\mathtt{q} := 0;$

$\mathtt{t}[0] := 0; \quad \mathtt{t}[1] := n;$

$\mathtt{at}[0] := 0; \quad \mathtt{mt}[0] := \{0\};$

for $\mathtt{k} := 1$ **to** $n-1$ **do**

  LinearSearch;

  if $\mathtt{q} < 0$ **then** *Reset*

  else *Update* **end**

end.

The fragment *LinearSearch* makes $\mathtt{q} := j$ where j is maximal with $t(j,k) < u(k)$:

LinearSearch:

  while $\mathtt{q} \geqslant 0 \wedge f(\mathtt{t}[\mathtt{q}], \mathtt{at}[\mathtt{q}]) > f(\mathtt{t}[\mathtt{q}], \mathtt{k})$

    do $\mathtt{q} := \mathtt{q} - 1$ **end**.

Fragment *Reset* handles the case $u(k) = 0$:

Reset:

  $\mathtt{q} := 0; \quad \mathtt{t}[1] := n;$

  $\mathtt{at}[0] := \mathtt{k}; \quad \mathtt{mt}[0] := \{\mathtt{k}\}.$

Fragment *Update* determines $\mathtt{w} = u(k) - 1$ and then restores the invariants:

Update:

  $\mathtt{w} := \min(n-1, g(\mathtt{at}[\mathtt{q}], \mathtt{k}));$

  if $\mathtt{w}+1 < \mathtt{t}[\mathtt{q}+1]$ **then**

    $\mathtt{t}[\mathtt{q}+1] := \mathtt{w}+1;$

    $\mathtt{mt}[\mathtt{q}] := \{\mathtt{at}[\mathtt{q}]\};$

  end;

  if $f(\mathtt{w}, \mathtt{at}[\mathtt{q}]) = f(\mathtt{w}, \mathtt{k})$ **then**

    $\mathtt{mt}[\mathtt{q}] := \mathtt{mt}[\mathtt{q}] \cup \{\mathtt{k}\};$

  end;

  if $\mathtt{w}+1 < n$ **then**

    $\mathtt{q} := \mathtt{q}+1; \quad \mathtt{t}[\mathtt{q}+1] := n;$

    $\mathtt{at}[\mathtt{q}] := \mathtt{k}; \quad \mathtt{mt}[\mathtt{q}] := \{\mathtt{k}\};$

  end.

We use $vf = 2(n - \mathtt{k}) + \mathtt{q}$ to analyse the complexity of the algorithm. Initially $vf = 2n - 2$. Since vf decreases both in the body of *LinearSearch* and in the body of the outer loop, the algorithm has linear time complexity. The resulting sets $\mathtt{M}[y] = M(y,n)$ are collected in

forall $j \in [0 \ldots \mathtt{q}]$ **do**

  forall $y \in [\mathtt{t}[j] \ldots \mathtt{t}[j+1] - 1)$

    do $\mathtt{M}[y] := \{\mathtt{at}[j]\}$ **end**;

  $\mathtt{M}[\mathtt{t}[j+1]-1] := \mathtt{mt}[j];$

end.

4. Concluding remarks

The algorithm for the simple feature transform presented in [6] can be obtained from the present one by removing the sets $\mathtt{mt}$, and just collecting the values

$\mathtt{at}[j]$. Although the present algorithm seems but a minor extension of the previous one, we had to rearrange the analysis completely to get it correct.

For applications in image processing, performance is usually a major issue. For that purpose, we decided to represent the sets $\mathtt{mt}[j]$ by means of two integer arrays $\mathtt{m}$ and $\mathtt{b}$ with the invariants $\mathtt{mt}[j] = \{\mathtt{m}[i] \mid \mathtt{b}[j] \leqslant i < \mathtt{b}[j+1]\}$ for $j \leqslant \mathtt{q}$, and $\mathtt{b}[0] = 0$ and $\mathtt{at}[j] = \mathtt{m}[\mathtt{b}[j]]$. Since array $\mathtt{mt}$ is only modified at index $\mathtt{q}$, this representation requires no shifting of values.

We have a prototype implementation in Java that determines the feature transform sets of a 3D image of $128 \times 128 \times 62$ voxels in 1.9 seconds on a Pentium 4 (3 GHz). The image is of blood vessels. It has 95940 voxels x for which $FT(x)$ contains more than one point. For these voxels, the average number of elements of $FT(x)$ is 2.2. In this case, our algorithm for the simple feature transform requires 0.365 seconds. Thus, a factor 5 is paid to represent and modify a huge number of very small sets.

References

[1] G. Borgefors, Distance transformations in arbitrary dimensions, Comput. Vision, Graphics, Image Process. 27 (1984) 321–345.

[2] H. Breu, J. Gil, D. Kirkpatrick, M. Werman, Linear time Euclidean distance transform algorithms, IEEE Trans. Pattern Anal. Machine Intell. 17 (1995) 529–533.

[3] L. Chen, H.Y.H. Chuang, A fast algorithm for Euclidean distance maps of a 2-d binary image, Inform. Process. Lett. 51 (1994) 25–29.

[4] M. Couprie, R. Zrour, Discrete bisector function and Euclidean skeleton, in: E. Andres, G. Damiand, P. Lienhardt (Eds.), Discrete Geometry for Computer Imagery 2005, in: Lecture Notes in Comput. Sci., vol. 3429, Springer, Berlin, 2005, pp. 216–227.

[5] P.-E. Danielsson, Euclidean distance mapping, Comput. Graphics, Image Process. 14 (1980) 227–248.

[6] W.H. Hesselink, M. Visser, J.B.T.M. Roerdink, Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform, in: C. Ronse, L. Najman, E. Decencière (Eds.), Mathematical Morphology: 40 Years On, Proc. 7th Internat. Symp. on Mathematical Morphology, April 18–20, Springer, Berlin, 2005, pp. 259–268.

[7] T. Hirata, A unified linear-time algorithm for computing distance maps, Inform. Process. Lett. 58 (1996) 129–133.

[8] C.R. Maurer Jr., R. Qi, V. Raghavan, A linear time algorithm for computing the Euclidean distance transform in arbitrary dimensions, IEEE Trans. Pattern Anal. Machine Intell. 25 (2003) 265–270.

[9] A. Meijster, J.B.T.M. Roerdink, W.H. Hesselink, A general algorithm for computing distance transforms in linear time, in: J. Goutsias, L. Vincent, D.S. Bloomberg (Eds.), Mathematical Morphology and its Applications to Image and Signal Processing, Proc. 5th Internat. Conf., Kluwer, Dordrecht, 2000, pp. 331–340.

[10] D.W. Paglieroni, A unified distance transform algorithm and architecture, Machine Vision Appl. 5 (1992) 47–55.

[11] A. Rosenfeld, J.L. Pfaltz, Sequential operations in digital picture processing, J. ACM 13 (1966) 471–494.